24 - Lexicographic permutations

September 9, 2018

A permutation is an ordered arrangement of objects. For example, 3124 is one possible permutation of the digits 1, 2, 3 and 4. If all of the permutations are listed numerically or alphabetically, we call it lexicographic order. The lexicographic permutations of 0, 1 and 2 are:
012 021 102 120 201 210
What is the millionth lexicographic permutation of the digits 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9?

```
In [2]: import math
```

I figured that $(1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10})$ has $1 * 9! = 326.880$ permutations. The index can be related to the permutation set in the following way:

$$\text{index} = 9! \cdot x_1 + 8! \cdot x_2 + \ldots + 0! \cdot x_n$$

where $x_1, x_2, \ldots x_n$ are indices. First we define the set of permutations as $S = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
We want to find maximum value of x_1 such that $9! \cdot x_1 \leq 1.000.000$ where x_1 is an integer. This gives:

$$x_1 = \left\lfloor \frac{1.000.000}{9!} \right\rfloor = 2 \implies S_3 = 2$$

This gives us the element that we have to select from S, namely S_{n+1}. This gives $S_3 = 2$. Now we remove 2 and store it in R. In S we are left with $\{0, 1, 3, 4, 5, 6, 7, 8, 9\}$. The remainder for the next iteration is found with $1.000.000 - 9! \cdot x_1$ which is 274.240. In the next iteration we find x_2:

$$x_2 = \left\lfloor \frac{274.240}{8!} \right\rfloor = 6 \implies S_7 = 7$$

Which gives $S_7 = 7$. We repeat this proces until the remainder is 0. The resulting permutation is found by adding our found $x_1, x_2, \ldots, x_{n-1}$, which are in R, and what is left in S.

```
In [29]: def find_permutation_at_index(S, index):
             R = []
             remainder = index
             while remainder > 0:
                 n = len(S)-1
                 index = remainder // math.factorial(n)
                 remainder -= index * math.factorial(n)
                 R.append(S[index])
                 del S[index]
             return R + S
```

```
In [33]: %%time
         find_permutation_at_index([0,1,2,3,4,5,6,7,8,9], 999999)
```

```
Wall time: 0 ns
```

```
Out[33]: [2, 7, 8, 3, 9, 1, 5, 4, 6, 0]
```

This algorithm works for any arbitrary permutation set. We can also use it, for example, on letters:

```
In [44]: for i in range(math.factorial(4)):
             print('i: {} = {}'.format(i, find_permutation_at_index(['A','B','C','D'], i)))
```

```
i: 0 = ['A', 'B', 'C', 'D']
i: 1 = ['A', 'B', 'D', 'C']
i: 2 = ['A', 'C', 'B', 'D']
i: 3 = ['A', 'C', 'D', 'B']
i: 4 = ['A', 'D', 'B', 'C']
i: 5 = ['A', 'D', 'C', 'B']
i: 6 = ['B', 'A', 'C', 'D']
i: 7 = ['B', 'A', 'D', 'C']
i: 8 = ['B', 'C', 'A', 'D']
i: 9 = ['B', 'C', 'D', 'A']
i: 10 = ['B', 'D', 'A', 'C']
i: 11 = ['B', 'D', 'C', 'A']
i: 12 = ['C', 'A', 'B', 'D']
i: 13 = ['C', 'A', 'D', 'B']
i: 14 = ['C', 'B', 'A', 'D']
i: 15 = ['C', 'B', 'D', 'A']
i: 16 = ['C', 'D', 'A', 'B']
i: 17 = ['C', 'D', 'B', 'A']
i: 18 = ['D', 'A', 'B', 'C']
i: 19 = ['D', 'A', 'C', 'B']
i: 20 = ['D', 'B', 'A', 'C']
i: 21 = ['D', 'B', 'C', 'A']
i: 22 = ['D', 'C', 'A', 'B']
i: 23 = ['D', 'C', 'B', 'A']
```

```
In [30]: for i in range(math.factorial(len([0,1,2]))):
             print('i: {} = {}'.format(i, find_permutation_at_index([0,1,2], i)))
```

```
i: 0 = [0, 1, 2]
i: 1 = [0, 2, 1]
i: 2 = [1, 0, 2]
i: 3 = [1, 2, 0]
i: 4 = [2, 0, 1]
i: 5 = [2, 1, 0]
```